# SUMAS PARCIALES
## DE ELEMENTOS CONSECUTIVOS

### Algorítmica

Grado en Ingeniería Informática
Universidad de granada

ADRA SANCHEZ RUIZ
CRISTINA GARRIDO AMADOR
MARIANA ORIHUELA CAZORLA
JUAN MANUEL CASTILLO NIEVAS
LUIS LIÑÁN VILLAFRANCA

29 de marzo de 2017, Granada

# Índice



# Índice de figuras



# Índice de tablas

# 1. Presentación del problema

La práctica que se nos ha asignado ha sido el cálculo de la suma parcial de elementos consecutivos. La cual tiene como objetivo encontrar la suma máxima entre los distintos componentes.

Para ello, hemos desarrollado un bucle anidado que va sumando el primer componente del vector con el segundo, guardando este valor en una variable. Seguidamente, suma ese valor con el tercero y comprueba si es mayor o no, si lo es, sigue sumando con el cuarto y así sucesivamente, sino, coge el último máximo y va sumando desde ese.

Esto lo hace para cada uno de los componentes del vector, hallando por tanto el valor máximo y siendo este el que se devuelve.

# 2. Pruebas realizadas

A la hora de hacer el cálculo empírico, hemos utilizado una macro de la siguiente forma:

```
#!/bin/bash

echo "" > $1.dat

for((i = 0; i < 100000; i += 1000)) do
    echo "Tamaño $i"
    echo `$@ $i` >> $1.dat
done
```

# 3. Resultados obtenidos

## 3.1. Algoritmo clásico

Como solución a este problema, proponemos primero un algoritmo clásico. Este algoritmo es simple:

INICIO Recorrer todos los elementos del vector menos el último

1 Coger el i-ésimo elemento del vector

2 Recorrer todos los elementos del vector a partir del elemento i+1 del punto anterior

3 En cada iteración, sumar a la variable suma el elemento actual. Si la suma es mayor que el máximo que teníamos hasta ahora, el máximo ahora es la variable suma.

BASE Devolver el máximo

Es fácil ver la eficiencia cuadrática que tiene este algoritmo. Tenemos un bucle for que recorre todos los elementos del vector menos el último (orden $O(n)$) y dentro de este bucle, tenemos otro bucle for que, en el peor de los casos, va a recorrer todos los elementos del vector menos el primero, con lo cual tenemos $O(n)$ otra vez. Como las demás sentencias son de $O(1)$, tenemos que:

$$\sum_{i=0}^{n} O(i) \in O(n^2)$$

### 3.1.1. Cálculo de la eficiencia empírica

| Vector size | Time (sec) |
|---|---|
| 1000 | 0.0023 |
| 2000 | 0.0079 |
| 3000 | 0.0137 |
| 4000 | 0.0232 |
| 5000 | 0.0306 |
| 6000 | 0.0423 |
| 7000 | 0.0567 |
| 8000 | 0.0738 |
| 9000 | 0.0937 |
| 10000 | 0.1149 |
| 11000 | 0.1413 |
| 12000 | 0.1646 |
| 13000 | 0.1934 |
| 14000 | 0.2240 |
| ... | |
| 85000 | 8.2522 |
| 86000 | 8.4425 |
| 87000 | 8.6385 |
| 88000 | 8.8398 |
| 89000 | 9.0516 |
| 90000 | 9.2474 |
| 91000 | 9.4298 |
| 92000 | 9.6841 |
| 93000 | 9.8733 |
| 94000 | 10.0894 |
| 95000 | 10.3090 |
| 96000 | 10.5203 |
| 97000 | 10.7451 |
| 98000 | 10.9694 |
| 99000 | 11.1872 |

Tabla 3.1: Tiempos obtenidos para el algoritmo clásico.

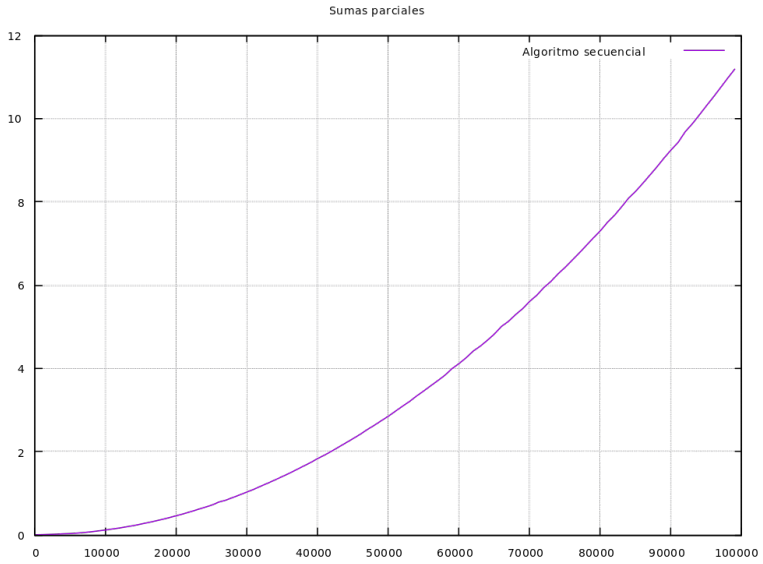


Figura 3.1: Gráfica empírica de la suma parcial de elementos realizada con un algoritmo secuencial.

### 3.1.2. Cálculo de la eficiencia híbrida

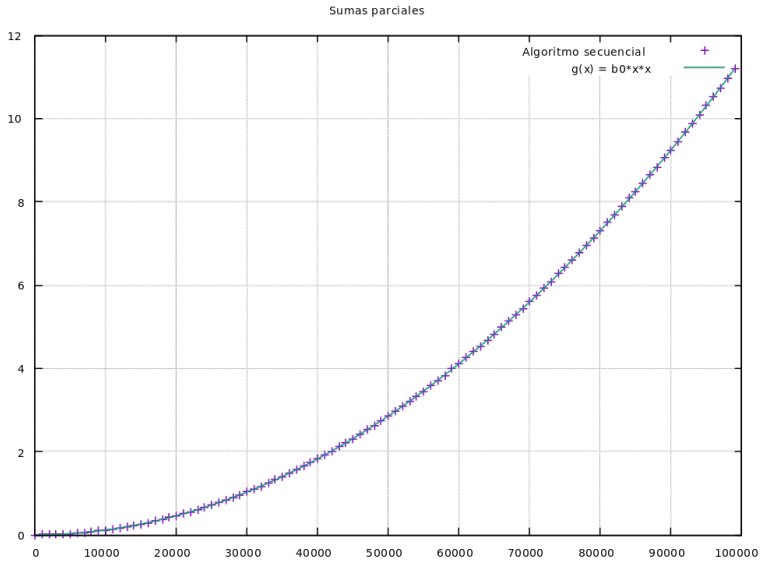


Figura 3.2: Gráfica híbrida de la suma parcial de elementos realizada con el algoritmo secuencial.

```
Final set of parameters   Asymptotic Standard Error
=======================   ==========================
a0 =         1.14155e-09  +/- 9.995e-13   (0.08755%)
a1 =          5.3884e-08  +/- 1.023e-07     (189.8%)
a2 =         0.000283546  +/- 0.00219       (772.5%)
```

Tabla 3.2: Ajustes y errores para la función $n^2$ ajustada al algoritmo secuencial.

Viendo la gráfica 3.1, se puede deducir que el algoritmo es de orden de eficiencia $O(n^2)$. Para comprobarlo, hemos utilizado gnuplot para realizar un ajuste híbrido (gráfica 3.2). Según los datos obtenidos (Tabla 3.2) el ajuste a este orden es casi perfecto (sólo un 0.08755 % de error al obtener la constante oculta $a0$ de $a0 * n^2 + a1 * n + a2$).

## 3.2. Algoritmo Divide y Vencerás

Para hacer este algoritmo un poco más eficiente, hemos realizado el siguiente algoritmo:

INICIO Dividir el vector por la mitad

1. Calcular el máximo del vector izquierda y del vector derecha de manera recursiva
2. Calcular la suma de las dos mitades, pues puede pasar que la suma máxima esté en mitad. Para ello:
   - 2.1 Recorrer el vector izquierda desde el final y sumar sus elementos para encontrar el máximo
   - 2.2 Recorrer el vector derecha desde el principio y sumar sus elementos para encontrar el máximo
3. Calcular el máximo entre el vector izquierda y el vector derecha
4. Calcular el máximo entre el máximo anterior y la suma de las dos mitades (apartado 2)
5. Devolver máximo

BASE El vector izquierda y el vector derecha son el mismo. Devolver el elemento que contiene.

Su eficiencia teórica es simple, pues tenemos que la función se llama dos veces a sí misma, pasándole como parámetro la mitad del vector, y luego tenemos dos bucles for que van a recorrer los elementos de los dos vectores, izquierda y derecha. Por lo tanto, tenemos $T(n) = 2 * T(\frac{n}{2}) + O(n)$, que como hemos visto en clase de teoría, se traduce en $O(n * log(n))$.

### 3.2.1. Cálculo de la eficiencia empírica

| Vector size | Time (sec) |
|---|---|
| 1000 | 0.0001 |
| 2000 | 0.0002 |
| 3000 | 0.0003 |
| 4000 | 0.0004 |
| 5000 | 0.0005 |
| 6000 | 0.0005 |
| 7000 | 0.0006 |
| 8000 | 0.0007 |
| 9000 | 0.0008 |
| 10000 | 0.0009 |
| 11000 | 0.0011 |
| 12000 | 0.0012 |
| 13000 | 0.0013 |
| 14000 | 0.0014 |
| ... | |
| 85000 | 0.0073 |
| 86000 | 0.0075 |
| 87000 | 0.0075 |
| 88000 | 0.0076 |
| 89000 | 0.0078 |
| 90000 | 0.0078 |
| 91000 | 0.0073 |
| 92000 | 0.0073 |
| 93000 | 0.0073 |
| 94000 | 0.0076 |
| 95000 | 0.0075 |
| 96000 | 0.0074 |
| 97000 | 0.0079 |
| 98000 | 0.0076 |
| 99000 | 0.0077 |

Tabla 3.3: Tiempos obtenidos para el algoritmo Divide y Vencerás.

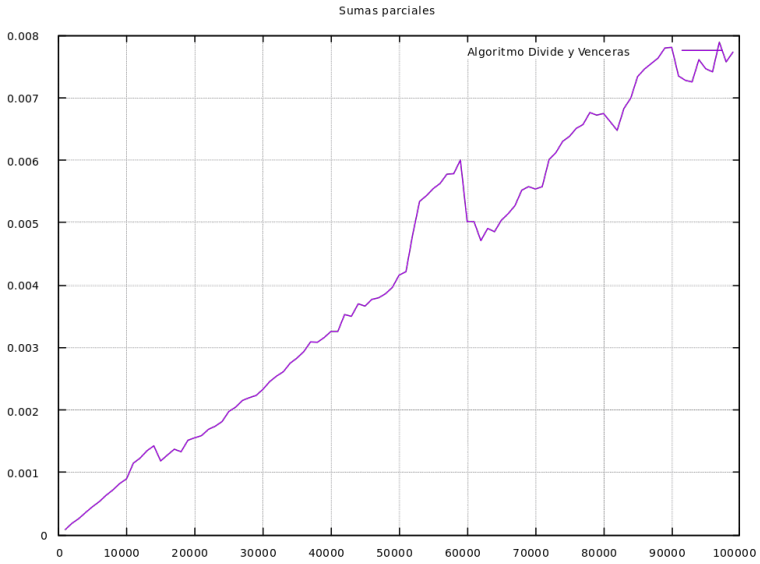


Figura 3.3: Gráfica empírica de la suma parcial de elementos realizada con un algoritmo Divide y Vencerás.

### 3.2.2. Cálculo de la eficiencia híbrida

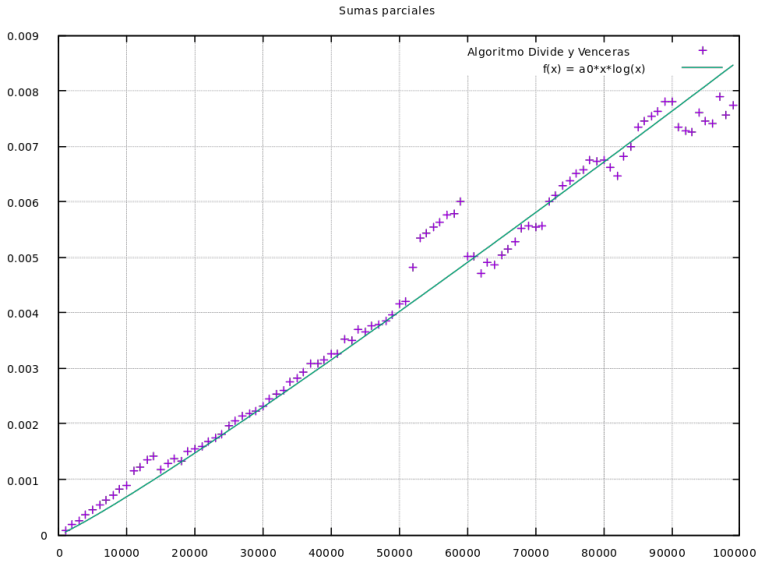


Figura 3.4: Gráfica híbrida de la suma parcial de elementos realizada con el algoritmo Divide y Vencerás.

```
Final set of parameters   Asymptotic Standard Error
=======================   ==========================
b0 =   0.00000000703773   +/- 1.079E-10    (1.533%)
b1 =        0.000296632   +/- 6.946E-05    (23.42%)
```

Tabla 3.4: Ajustes y errores para la función $n * log(n)$ ajustada al algoritmo Divide y Vencerás.

Viendo la gráfica 3.3, se puede deducir que el algoritmo es de orden de eficiencia $O(n * log(n))$. Para comprobarlo, hemos utilizado gnuplot para realizar un ajuste híbrido (gráfica 3.4). Según los datos obtenidos (Tabla 3.4) el ajuste a este orden es casi perfecto (sólo un 1.533 % de error al obtener la constante oculta $b0$ de $b0 * n * log(n) + b1$).

## 3.3. Comparación

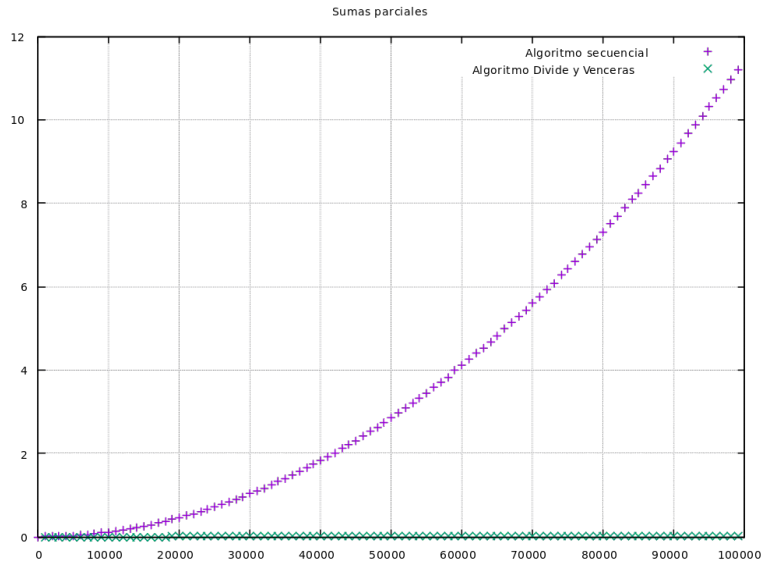


Figura 3.5: Comparación de la eficiencia de dos algoritmos.

Como podemos observar en la gráfica 3.5, no es nada comparable la salida de tiempos del algoritmo clásico con el Divide y Vencerás para los mismos tamaños.